ALGONQUIN COLLEGE

PLATYPUS LANGUAGE SPECIFICATION[1]

General View

Grammar, which knows how to control even kings . . .
—Molière, *Les Femmes Savantes* (1672), Act II, scene vi

A context-free grammar is used to define the lexical and syntactical parts of the **PLATYPUS** language and the lexical and syntactic structure of a **PLATYPUS** program.

1. The PLATYPUS 3.0 Lexical Specification

1.1. White Space

White space is defined as the ASCII space, horizontal and vertical tabs, and form feed characters, as well as line terminators. White space is discarded by the scanner.

<white space> → *one of* { `SPACE TAB FF NL CR NLCR` }

1.2. Comments

PLATYPUS supports only single-line comments: all the text from the ASCII characters **%%** to the end of the line is ignored by the scanner.

<comments> → **%%** { sequence of ASCII chars }

1.3. Variable Identifiers

[1] Adapted from resources developed by Prof. Svillen Ranev (Algonquin College, 2019)

The following variable identifier (VID) tokens are produced by the scanner: AVID_T and SVID_T.

<variable identifier> → AVID_T | SVID_T

1.4. Keywords

The scanner produces a single token: **KW_T**. The type of the keyword is defined by the attribute of the token (the index of the keywordTable []).

1.5. Integer Literals

The scanner produces a single token: **INL_T** with an integer value as an attribute.

1.6. Floating-point Literals

FPL_T token with a real decimal value as an attribute is produced by the scanner.

1.7. String Literals

STR_T token is produced by the scanner. The attribute is the string literal offset (currentToken.attribute.str_offset) from the beginning of the string literal buffer (stringLiteralTable->string).

1.8. Separators

<separator> → *one of* { `( ){ } , ;` }

Seven different tokens are produced by the scanner - **LPR_T**, **RPR_T**, **LBR_T**, **RBR_T**, **COM_T**, **EOS_T**.

1.9. Operators

<arithmetic operator> → *one of* { `+, -, *, /` }

A single token is produced by the scanner: **ART_OP_T**. The type of the operator is defined by the attribute of the token.

<string concatenation operator> → ++

A single token is produced by the scanner: **SCC_OP_T**.

<relational operator> → *one of* { `>, <, ==, !=` }

A single token is produced by the scanner: **REL_OP_T**. The type of the operator is defined by the attribute of the token.

<logical operator> → *one of* { `.AND., .OR., .NOT.` }

A single token is produced by the scanner: **LOG_OP_T**. The type of the operator is defined by the attribute of the token.

<assignment operator> → =

A single token is produced by the scanner: **ASS_OP_T**.

2. The PLATYPUS Syntactic Specification

2.1. PLATYPUS Program

2.1.1. Program

<program> → **MAIN** {<opt_statements>}

First Set

FIRST(<program>) = { KW_T(**MAIN**) }

Optional Statements:

<opt_statements> → <statements> | ϵ

First Set

TODO_01

2.1.2. Statements

<statements> → <statement> | <statements> <statement>

- **PROBLEM DETECTED: Left recursion (SOLVED for you here):**

New Grammar

<statements> → <statement><statementsPrime>
<statementsPrime> → <statement><statementsPrime> | ϵ

First Set

TODO_02

2.2. Statements

<statement> → <assignment statement> | <selection statement> | <iteration statement>
| <input statement> | <output statement>

First Set

TODO_03

2.2.1. Assignment Statement

<assignment statement> → <assignment expression>;

First Set

TODO_04

2.2.2. Assignment Expression

<assignment expression> → AVID = <arithmetic expression> | SVID = <string expression>

First Set

TODO_05

2.2.3. Selection Statement (if statement)

<selection statement> → IF <pre-condition> (<conditional expression>)
THEN { <opt_statements> }
ELSE { <opt_statements> } ;

First Set

TODO_06

2.2.4. Iteration Statement (the loop statement)

<iteration statement> → WHILE <pre-condition> (<conditional expression>)
DO { <statements>};

First Set

TODO_07

<pre-condition> → TRUE | FALSE

First Set

TODO_08

2.2.5. Input Statement

<input statement> → READ (<variable list>);

First Set

TODO_09

Variable List:

<variable list> → <variable identifier> | <variable list>,<variable identifier>

- **PROBLEM DETECTED: Left recursion:**

New Grammar
TODO_10

First Set
TODO_11

Variable Identifier:

<variable identifier> → AVID_T | SVID_T

First Set
TODO_12

2.2.6. Output Statement

<output statement> → WRITE (<opt_variable list>); | WRITE (STR_T);

- **PROBLEM DETECTED: Left factoring (SOLVED for you here):**

New Grammar
<output statement> → WRITE (<output statementPrime>); **<output statementPrime>** → <opt_variable list> | STR_T

First Set
TODO_13

Optional Variable List:

<opt_variable list> → <variable list> | ϵ

First Set
TODO_14

- **Note:** In some cases, the grammar may be transformed to predictive grammar without applying the general rule. For example, the grammar above can be rewritten as follows.

- **Rewriting the grammar:**

New Grammar
<output statement> → OUTPUT (<output list>); **<output_list>** → <opt_variable list> | STR_T

First Set

TODO_15

2.3. Expressions

2.3.1. Arithmetic Expression

<arithmetic expression> → <unary arithmetic expression> | <additive arithmetic expression>

First Set

TODO_16

Unary Arithmetic Expression:

<unary arithmetic expression> → - <primary arithmetic expression>
| + <primary arithmetic expression>

First Set

TODO_17

Additive Arithmetic Expression:

<additive arithmetic expression> →
<additive arithmetic expression> + <multiplicative arithmetic expression>
| <additive arithmetic expression> - <multiplicative arithmetic expression>
| <multiplicative arithmetic expression>

- **PROBLEM DETECTED: Left recursion:**

New Grammar

TODO_18

First Set

TODO_19

Multiplicative Arithmetic Expression:

<multiplicative arithmetic expression> →
<multiplicative arithmetic expression> * <primary arithmetic expression>
| <multiplicative arithmetic expression> / <primary arithmetic expression>
| <primary arithmetic expression>

- **PROBLEM DETECTED: Left recursion:**

New Grammar

TODO_20

First Set

TODO_21

Primary Arithmetic Expression:

<primary arithmetic expression> → AVID_T | FPL_T | INL_T
| (<arithmetic expression>)

First Set

TODO_22

2.3.2. String Expression

<string expression> →
<primary string expression> | <string expression> ++ <primary string expression>

- **PROBLEM DETECTED: Left recursion:**

New Grammar

TODO_23

First Set

TODO_24

Primary String Expression:

<primary string expression> → SVID_T | STR_T

First Set

TODO_25

2.3.3. Conditional Expression

<conditional expression> → <logical OR expression>

First Set

TODO_26

Logical OR Expression:

<logical OR expression> → <logical AND expression>
| <logical OR expression> **.OR.** <logical AND expression>

- **PROBLEM DETECTED: Left recursion:**

New Grammar

TODO_27

First Set

TODO_28

Logical AND Expression:

<logical AND expression> → <logical NOT expression>
| <logical AND expression> **.AND.** <logical NOT expression>

- **PROBLEM DETECTED: Left recursion:**

New Grammar

TODO_29

First Set

TODO_30

Logical NOT Expression:

<logical NOT expression> → **.NOT.** <relational expression>
| <relational expression>

First Set

TODO_31

2.3.4. Relational Expression

<relational expression> →
<relational a_expression> | <relational s_expression>

First Set

TODO_32

Relational Arithmetic Expression:

<relational a_expression> →
<primary a_relational expression> == <primary a_relational expression>
| <primary a_relational expression> != <primary a_relational expression>
| <primary a_relational expression> > <primary a_relational expression>
| <primary a_relational expression> < <primary a_relational expression>

- **PROBLEM DETECTED: Left factoring:**

New Grammar

TODO_33

First Set

TODO_34

Relational String Expression:

<relational s_expression> →

| <primary s_relational expression> == <primary s_relational expression>
| <primary s_relational expression> != <primary s_relational expression>
| <primary s_relational expression> > <primary s_relational expression>
| <primary s_relational expression> < <primary s_relational expression>

- **PROBLEM DETECTED: Left factoring:**

New Grammar

TODO_35

First Set

TODO_36

Primary Arithmetic Relational Expression:

<primary a_relational expression> → AVID_T | FPL_T | INL_T

First Set

TODO_37

<primary s_relational expression> → <primary string expression>

First Set

TODO_38

Good luck with Assignment 3!